International Journal of Information Security
https://doi.org/10.1007/s10207-019-00465-8

SPECIAL ISSUE PAPER

Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability

Shufan Zhang[1] · **Lili Wang**[1,2] · **Hu Xiong**[1]

Abstract
Blockchain-enabled voting (BEV) systems have emerged as the next generation of modern electronic voting (e-voting) systems, because the immutable property of the blockchain has made itself a perfect distributed ballot box. Further, recent investigations have utilized the smart contract to build a decentralized autonomous voting application over blockchain. We identify nine critical desiderata, such as scalability, verifiability, and robustness, that a BEV system *can* and *should* achieve. However, we find that existing BEV systems violate at least one of the nine desiderata. In light of this deficiency, we propose a novel BEV system, named Chaintegrity, that fulfills all the specified desiderata. In addition, to make our system more cost-effective, we also propose a hybrid data structure which combines the counting Bloom filter and the Merkle hash tree for fast authentication. To enhance robustness, we as well introduce the code-voting technique as a component in our system. Our empirical results also show that our system achieves high efficiency and enjoys low computational and communication overhead.

Keywords E-voting protocol · Robustness · Large scale · Universal verifiability · Blockchain · Smart contract

1 Introduction

Advances in information and communication technology (ICT) over the decades have broken geographical hindrance, allowing voters to make democratic decisions remotely and ubiquitously via the Internet. This manner, called remote electronic voting (REV), has brought convenience to both the voters and the election managers, helping to reduce the human cost. But at the same time, it is no news these days to hear remote absentee ballot tampering in a voter fraud incident. Some latest examples are the ballot tampering scandal in the 2019 North Carolina elections [55] and the server wiping in the 2017 Georgia elections [6], just to name a few. As recent studies [3,33] revealed, the vulnerabilities of the centralized ballot storage in REV systems are exploited to rig elections, and this may arouse among the voters the crisis of confidence in the authorities.

The advent of the blockchain technology is widely expected to solve this problem and disrupt both the traditional in-person voting and the modern electronic voting (e-voting) approaches [24,39]. Blockchain, as the backbone technology of Nakamoto's Bitcoin system [32], is an immutable and transparent ledger distributed on the peer-to-peer network. It deploys the consensus algorithm [34] to address the inconsistency problem[1] in the distributed systems. Any changes in the data stored on one distributed node will immediately be detected by other nodes in the network. In the context of REV, this property promises unalterable ballot storage, provides a clear record of the vote cast, and makes blockchain a perfect ballot box to eliminate potential voter fraud.

The application of blockchain in REV is gaining momentum. This has entrenched a line of researches these years—blockchain-enabled voting (BEV) systems. Some prominent works also employ smart contract [69], the self-executed on-chain scripts, to build decentralized autonomous voting applications, which can decrease the human factors as

✉ Hu Xiong
xionghu.uestc@gmail.com

[1] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu, China

[2] School of Network Security, Chengdu University of Technology, Chengdu, China

[1] It is also known as the double-spend problem, in terms of cryptocurrency and fintech.

Published online: 03 September 2019

many as possible. In this paper, we revisit the state-of-the-art investigations on BEV systems but surprisedly find that no common set of system requirements have yet arrived. What's more, some works claimed to achieve the properties which are impractical on blockchain. This motivates us to start a second glance upon the properties that a BEV system *can* and *should* satisfy. We also propose a novel system to fulfill all the proposed requirements. Specifically, the prime technical contributions in this paper are as follows:

1. We formulate the current problem in large-scale remote voting and explain the failure of existing blockchain-enabled voting systems of simultaneously satisfying cost-efficacy, verifiable, and robustness. For the first time in the literature, we identify the maximum set of properties that the large-scale blockchain voting can achieve, and particularly prove the impossibility of receipt-freeness in blockchain voting.
2. To cope with the existing problem, we propose Chaintegrity—a novel, clear, and robust system to meet all the requirements, which exploits the blind signature scheme and the distributed homomorphic encryption scheme, tailors and adapts them for large-scale elections. To make our voting system more cost-friendly, we also propose a hybrid approach which combines the counting bloom filter and Merkle hash tree to break the bottleneck. The bloom filter and the Merkle hash tree are widely used in the blockchain and its applications, but they are not originally designed for the e-voting system. We discuss their usage in distributed blockchain-enabled e-voting system and analyze the trade-off between the efficiency promotion and the tolerable false positive rate. Apart from this, we also discuss the problem in real-world deployment and introduce the code-voting technique as a component to provide dispute resolution functions.
3. We prove that our system fulfills all the requirements that a blockchain-enabled large-scale voting system should achieve. We also use the experimental results to analyze the efficiency of our system. The experiments and evaluation demonstrate the cost-effectiveness of Chaintegrity in practice.

1.1 Related works

Initial attempts on designing e-voting protocols over blockchain spawned cryptocurrency incentive systems. Inspired by a lottery protocol [10], Zhao and Chan [71] proposed a protocol for Bitcoin voting, which allows n funded voters to vote for one of two candidates by transferring Bitcoin. In their work, ballots are concealed by random numbers which are generated distributedly. Voters commit in an off-chain manner to both the masked ballot and the random number and then reveal their commitments over Bitcoin. However, their protocol provides no real anonymity for voters, due to the pseudonymity nature of Bitcoin, and is restricted by approval elections (single "Yes/No" voting type). These two drawbacks are, respectively, improved in [56,57]. In [56], Takabatake et al. proposed the utilization of Zerocoin, a laundering extension that provides anonymity for Bitcoin, to construct voting systems. And in [57], Tian et al. proved that Bitcoin voting is not restricted to the binary type by exploiting Zhao and Chan's protocol [71] as a subroutine. Other monetary incentive e-voting systems, like [7,11], utilize different cryptographic primitives such as anonymous Kerberos and Circle Shuffle in their constructions, which has enriched and extended the properties (e.g., end-to-end verifiability and efficiency) of blockchain voting.

As a common feature of monetary incentive e-voting systems, honest voters will get refunded or rewarded while the malicious ones are penalized, when an election is over. Though seems reasonable, this mechanism in fact encourages deviation behaviors and results in lower voter turnout. The reasons are split into two halves. On the first half, voters may not pay the huge transaction fee[2] in advance for a potential reward. On the other, with promised higher benefits from vote-buyers, voters are more likely to undertake the financial penalty. Furthermore, due to the huge financial cost, monetary incentive systems are impractical for large-scale national elections [36]. Hence some e-voting systems turn to regard blockchain as an immutable and transparent ballot box [39]. In former times, verifiable voting systems [53] rely on complicated cryptographic constructions, while nowadays, blockchain may disrupt this limitation [36]. Chaieb et al. [16] and Pawlak et al. [50] independently argued that blockchain can and should be used to achieve universal verifiability and transparent auditability in e-voting systems. Some commercial startups, like TIVI [58], FollowMyVote [30] and Agora [2], also launch their BEV projects and implement the blockchain as a public ballot box. They claim that their systems are scalable, anonymous, verifiable and publicly accessible. However, lack of concrete system constructions in their technical documents and white papers makes it hard to evaluate whether or not their systems achieve commensurate properties as claimed.

With the development in blockchain technology, smart contract [27] takes the BEV a step further. Smart contracts are digital contracts that the clauses are enforced autonomously, which can be utilized to hold elections without the arbitration of a trusted administrator. Based on Open Vote Network [35], McCorry et al. [41] proposed a boardroom (i.e., small scale) self-tallying e-voting protocol, which is implemented as a smart contract on Ethereum blockchain. Though provides

[2] So far, the highest transaction fee is up to 55.16 USD, which is reached at the end of 2017: https://bitinfocharts.com/comparison/bitcoin-transactionfees.html.

maximum voter privacy, their e-voting system is not robust enough to defend Denial-of-Service (DoS) attack. Moreover, as the limitation of Ethereum platform, the voting scale is restricted to 50–60 voters. Recently, sharding-style [22,38] and EOS-style [26] blockchains are proposed, which break through the hindrance of low transaction rate and block generation rate. Yu et al. [67] suggested to use smart contract to build a secure voting system with no concerns about blockchain itself (i.e., platform-independence). Their system, based on linkable ring signature and homomorphic encryption, is proved to be scalable and can defend a spectrum of known attacks.

1.2 Roadmap

The remainder of this paper is organized as follows. In Sect. 2, we specify the large-scale e-voting problem in the context of blockchain application and evaluate related works. In Sect. 3, we introduce some background knowledge, such as the blockchain, the smart contract, the blind signature, the (distributed) homomorphic encryption and some special encoding techniques. Section 4 presents the model of our election system, including the participating entities, the system overview, the trust and threat models, and the design goals. Section 5 describes the basic framework of Chaintegrity, and Sect. 6 delineates some extended components which promote the efficiency and the robustness. Section 7 analyses the security properties of Chaintegrity. The experimental results are discussed in Sect. 8. Finally, we conclude this paper in Sect. 9.

2 Problem statement

In this section, we set the large-scale e-voting problem and propose nine key properties as evaluation criteria. We then use them to discuss prior attempts to deal with the problem and their failure to build such a system to satisfy the properties simultaneously. We also point out the failure in prior attempts to achieve receipt-freeness via blockchain. The comparing results motivate us to propose a novel solution.

2.1 The large-scale electronic voting problem

Consider that several candidates are chasing for one position and a large number of voters participate in the election. The voter can cast his/her supporting ballot for one candidate but can only vote once. As a result, the candidate who obtains the votes from the majority will win the competition.

Different parties have different requirements in this scenario, which make the voting problem difficult. As for a voter Alice, she wants to keep her choice secret because she does not want anyone knows that she voted for Carol, not her friend Bob. Alice also intends to ensure that her choice was truthfully recorded and tallied. As for a candidate Bob, he would expect that no partial result can be known to all prior to the opening day, since any temporary precedence of his competitors may attract neutral voters. As for the voting authorities or election holders, they need a cost-effective and affordable system. As well, they need the system to be robust and auditable so that they need not add cost to resolve disputes.

We conclude the nine properties that a large-scale blockchain enabled voting system should satisfy from the stated real-world problem and the reviews of blockchain voting [4,36] and traditional verifiable voting [53,59]. Note that, since the properties are derived from the requirements of different parties, some properties somewhat overlap with others, while some of them somewhat conflict with others. We list their definitions as follows and use them as the evaluation criteria to assess the prior works and our proposed system.

- C1 *Scalability* The voting system can support a large scale election, which requires high system concurrency and process efficiency.
- C2 *Privacy-enhancement* Voter's privacy requires an everlasting obfuscation in voter–vote relationship which cannot be distinguished by external observations. Particularly, given any specific vote, it is probabilistically impossible to find out which voter cast that vote.
- C3 *Universal verifiability* (*a.k.a. transparent auditability*) Universal Verifiability allows the public, who even do not have the right to vote, can also download the transcript of election and audit the completeness of an election.
- C4 *End-to-end verifiability* End-to-end verifiability means a voter can check the integrity of his/her vote in any phase of an election, which includes *cast as intended*, *recorded as cast* and *counted as recorded*.
- C5 *Vote-and-go* A voter can go off-line after his ballot is casted, where no further action in tallying needs voter's involvement. Compared with previous traditional e-voting protocols [31,47] and blockchain-based voting protocols [41], voter neither needs to trigger tallying nor participates in whole tallying process.
- C6 *Unreusability* Ballots from the same legitimate voter will not be counted except for his/her first ballot. This property thwarts voters' abuses and replay attacks.
- C7 *Affordability* The transaction fee should be affordable to both the voters and the election holders.
- C8 *Fairness* Fairness means the voting process is fair to everyone, where no one can break the protocol to get any partial results before final tallying.
- C9 *Robustness* A robust system can tolerate a certain degree of failure and adversarial inputs during execution. Furthermore, the robustness requires the ability of the system to run the election when confronting a minority of

Table 1 A comparative evaluation of blockchain enabled voting systems

Schemes	Evaluation criteria									Decentralisation	Authentication	Platform	Voting type
	C1	C2	C3	C4	C5	C6	C7	C8	C9				
Zhao et al. [71]	×	×	✓	✓	×	✓	×	×	×	No administrator	Plain	Bitcoin	Single
Takabatake et al. [56]	×	✓	✓	✓	✓	✓	×	✓	×	Single administrator	Off-chain authentication	Bitcoin + Zerocoin	Multiple
Bistarelli et al. [11]	×	✓	✓	✓	✓	✓	×	×	×	AS + TDS	Anonymous Kerberos	Bitcoin	Multiple
SHARVOT [7]	×	✓	✓	✓	✓	✓	×	✓	×	Single administrator	Trusted dealer	Bitcoin	Multiple
FollowMyVote [30]	–	✓	✓	✓	–	–	–	×	–	Not mentioned	Trusted delegate	BitShares	Multiple
TIVI [58]	✓	✓	✓	✓	✓	–	–	✓	–	Not mentioned	Not mentioned	Not mentioned	Single
Agora [2]	✓	✓	✓	✓	✓	×	✓	✓	×	Multi-administrators	Not mentioned	Customized blockchain	Not mentioned
VYV [16]	✓	✓	✓	✓	✓	✓	×	×	×	Single administrator	Plain	Ethereum	Multiple
McCorry et al. [41]	×	✓	×	✓	×	✓	×	✓	×	No administrator	Plain	Ethereum	Single
Yang et al. [66]	×	✓	✓	✓	×	✓	×	✓	×	Single administrator	Off-chain authentication	Ethereum	Multiple
Yu et al. [67]	✓	✓	✓	✓	✓	✓	✓	×	×	Single administrator	Ring signature	Platform-independent	Multiple
Our scheme	✓	✓	✓	✓	✓	✓	✓	✓	✓	Multi-administrators	Blind signature	Platform-independent	Multiple

✓, implemented; ×, not implemented; –, no explicit information; AS, authentication server; TDS, token distribution server

dishonest election authorities. As pointed out in [53], the misbehavior includes the rejection of tellers to decrypt ciphertexts, failure of mix servers to operate, etc.

2.2 Evaluation of prior works

By using the above criteria (C1–C9), we evaluate the prior works that are mentioned in Sect. 1.1. An intuitional comparison is presented in Table 1. In this table, we objectively compare the achievement of the properties and other aspects including the decentralisation degree, the authentication manner, the used blockchain platform and the used voting type.

Among all prior works, we notice that none of them really satisfy *robustness* as specified. Most systems (e.g., [41]) cannot defend Denial-of-Service (DoS) attacks and hence cannot tolerate partial failure. The nearest works are Agora [2] and Yu et al.'s [67]. However, we cannot find how Agora defends double-vote attacks from its whitepaper. Likely, we observe that the single tallier in Yu's system can readily refuse to decrypt the ciphertext-form tallying result. Thus, both Agora and Yu's system are not robust. And at the same time, Agora loses *unreusability*, and Yu's system violates *fairness*.

Another observation is that most systems cannot satisfy *affordability* and *scalability*. These blockchain-enabled voting systems are built upon either Bitcoin or Ethereum and need to afford a large amount of transaction fee (or gas price in terms of Ethereum). The capability to accommodate millions of voters is also restricted by the blockchain platform and the protocols themselves. Although researchers have tried their best to adapt their protocols to the blockchain platforms, their systems still fail to gain scalability. Rather, some resent designs [2,16,67] leverage consortium blockchains or EOS/sharding-style blockchains to handle the scalable problems. We follow their designs in our system.

With respect to (universal or end-to-end) *verifiability*, the essential goal of introducing blockchain to e-voting, almost all systems have achieved it. We also notice that all evaluated systems have used different mechanisms to provide *everlasting privacy-enhancement* for voters. An exempt is Zhao and Chan's lottery-based voting protocol [71]. In their protocol, the voters need to claim for reimbursement after the election. The evaluation result motivates us to build such a system that satisfies all the properties. Besides, we intend to mitigate the trust to centralized voting authorities and use a elaborative coding approach to support multiple voting type.

2.3 On the impossibility of receipt-free blockchain voting

Unfortunately, blockchain is not a panacea for all existing voting systems. A stronger notion of privacy, called receipt-freeness, which prevent a voter sells his/her vote to a potential vote-buyer, cannot be achieved. This is partially because that, receipt-freeness requires the authorized entity to add some *voter unknown* randomness to the encrypted ballot, whereas the smart contract cannot generate a *secret* random number.[3] On the other hand, the untappable channels, which appear in many receipt-free protocols, are extremely impractical in the distributed environment. Accordingly, some prior BEV protocols [2,16,67] are flawed.

Hence, we do not include receipt-freeness into our evaluation criteria. Receipt-freeness *is* important to large-scale

[3] The smart contract is the script distributedly executed on *all* validation nodes. Thus, as regards the consistency of the blockchain, it is impossible for every node *privately* chooses the *same* random number.

remote elections, whereas in some other real-world scenarios [1], voters are more care about whether their preferences are truthfully recorded and tallied. Adida [1] call these scenarios low-coercion elections, which are specifically held in the settings like student government and open-source software communities. Out of considering the features of blockchain, we believe that a blockchain-enabled e-voting system is more suited to these settings, and we intend to build the system for the mentioned purposes.

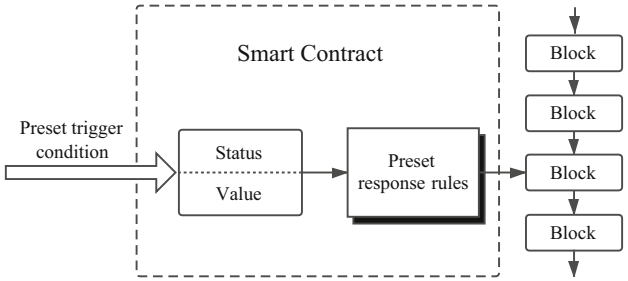

Fig. 1 The working principal of the smart contract

3 Building blocks

In this section, we introduce the blockchain, the smart contract, the blind signature, the homomorphic encryption, the encoding and decoding approaches for this work. For the succinctness of description, we only review the function notions of the blind signature and the homomorphic encryption and leave the concrete schemes to the appendices.

3.1 Blockchain and smart contract

Nakamoto introduced blockchain as a backbone of the Bitcoin system to solve the double-spend problem. The blockchain is a chain of blocks ordered by time, where the hash value of a previous block is stored in the next block. Any changes happen in an early block will exert an avalanche effect on changing hash value of all later blocks. Accordingly, any attempts of private interpolation to an early block are impractical. The blockchain was originally designed to store transaction information, while its effect has gone beyond the economic field since the introduction of the OP_RETURN instruction in Bitcoin. It allows the users to embed 40 bytes of data in a transaction, which makes the blockchain a trusted distributed database. The *trust* toward a blockchain means that the user has confidence in the decision made by a large proportion of nodes in the distributed system.

Without smart contract, blockchain is merely a tamper-resistant database; smart contract extend and leverage blockchain technology [65]. Smart contract is a family of predefined protocols which are deployed on blockchain. In other names, it is also called *chaincode* [5,15]. As illustrated in Fig. 1, smart contract works like finite state machine, where preset status and rule codes are encapsulated. When a trigger condition is met, smart contract examines the inside and outside rules, executes to respond according to the coming request. The handled responses (usually in form of status/value pairs) are finally written into blockchain.

3.2 Digital signature and blind signature

The digital signature is used to verify the authenticity of messages. Then the concept of blind signatures was firstly introduced by Chaum [17]. As a special digital signature, the content of the message is blinded before it is signed.

Blind signature is widely used for anonymous authentication [60,61] and in applications such as electronic voting and electronic cash/payment systems [51] since it provides an efficient approach to protect users' privacy [62,63]. Different from normal signature schemes, user and signer are separated in blind signature schemes. By adopting blind signature, user can obtain a wanted signature from the signer without leaking any knowledge of the message to be signed. The blind signature consists of following algorithms, and the details are described in "Appendix A.2."

- *Key generation* $(\mathsf{sk}_\mathsf{B}, \mathsf{pk}_\mathsf{B}) \leftarrow \mathit{BlindGen}(1^k)$ where k is the security parameter of the system. This function generates the signer's secret key sk_B and corresponding public key pk_B.
- *Blind* $m' \leftarrow \mathit{Blind}(m, r, \mathsf{pk}_\mathsf{B})$ takes the message m, the blind factor r and the signer's public key sk_B as the input, outputs the blind message m'.
- *Sign* $d \leftarrow \mathit{BlindSign}_{\mathsf{sk}_\mathsf{B}}(m')$. This function signs the blind message m' with the signer's secret key sk_B, generating the signature of the blind message d.
- *Unblind* $y \leftarrow \mathit{Unblind}(d, r)$ takes the signature of the blind message d and blind factor r as the input, outputs the message's signature y.
- *Verification* $1/0 \leftarrow \mathit{BlindVerf}_{\mathsf{pk}_\mathsf{B}}(y, m)$. This function inputs the message m, the message's signature y and the signer's public key sk_B, returns 1 if y is the signer's signature of message m, 0 otherwise.

3.3 Homomorphic encryption and threshold Paillier encryption

Homomorphic encryption allows calculations on ciphertexts to generate the encrypted result, and the decrypted result matches the result of the operations just like they had been executed on the plaintexts. As a common partially homomorphic cryptosystem, the Paillier encryption provides the additive homomorphic property. Let ℓ denotes the message, and homomorphic property holds $\mathit{Enc}(\ell_1) \cdot \mathit{Enc}(\ell_2) =$

$Enc(\ell_1+\ell_2)$, which make it possible to calculate the encryption of $\Sigma_{i=1}^{\rho}\ell_i$ without exposing plaintexts.

Threshold Paillier encryption allows multiple parties to share the secret together. Secret can only be decrypted if more than the threshold number of parties agree to decrypt. The Threshold Paillier encryption consists of several following algorithms, and the details are described in "Appendix B."

- *Key generation* $(\{\mathsf{sk}_{\mathsf{Pa}_1}, \mathsf{sk}_{\mathsf{Pa}_2}, \ldots, \mathsf{sk}_{\mathsf{Pa}_n}\}, \mathsf{pk}_{\mathsf{Pa}}) \leftarrow PaillierGen(1^k)$ where k is the security parameter of the system. This function generates the user's secret key $\{\mathsf{sk}_{\mathsf{Pa}_1}, \mathsf{sk}_{\mathsf{Pa}_2}, \ldots, \mathsf{sk}_{\mathsf{Pa}_n}\}$ and corresponding public key $\mathsf{pk}_{\mathsf{Pa}}$.
- *Encryption* $C \leftarrow Enc_{\mathsf{pk}_{\mathsf{Pa}}}(\ell)$ where ℓ is the plaintext of the ballot result, which holds $\ell \in \mathbb{Z}_n$. And C denotes the ciphertext, which is encrypted with $\mathsf{pk}_{\mathsf{Pa}}$.
- *Decryption* $C_i \leftarrow Dec_{\mathsf{sk}_{\mathsf{Pa}_i}}(C)$ where $\mathsf{sk}_{\mathsf{Pa}_i}$ is a share of the secret key. And each party execute this function to get the partial decryption C_i.
- *Combination* $\ell \leftarrow Com_{\mathsf{pk}_{\mathsf{Pa}}}(\prod)$. This function combine at least $t+1$ partial decryption $\prod = \{C_1, C_2, \ldots, C_{t+1}\}$ and decrypt the ballot result as ℓ.
- *Non-interactive zero-knowledge proof of membership* $\{v_j, e_j, u_j\}_{j\in P} \leftarrow PoK_{mem}(C, L)$ where C is the ciphertext, and L is the set of plaintext. This function generates a transcript $\{v_j, e_j, u_j\}_{j\in P}$ to prove that the corresponding plaintext of C lies in the set L.
- *Non-interactive zero-knowledge proof of correctness of partial decryption* $(R_1, R_2, e', z) \leftarrow PoK_{cor}(C, C_i)$ where C is the ciphertext, and C_i is the partial decryption. This function generates a transcript (R_1, R_2, e', z) to prove that C_i is the partial decryption of ciphertext C.

3.4 The counting Bloom filter and the Merkle hash tree

The counting Bloom filter [29] is a special variation of the Bloom filter. It is an efficient probabilistic data structure which supports membership queries with low temporal and spacial overheads. As shown in Fig. 2, a counting Bloom filter cBF for the set S supports the following 4 types of functions.

- *Initiation* This algorithm takes as input two integers $m, k \in N$. It samples k different hash functions $H_1, H_2, \ldots, H_k$, where each $H_i: S \rightarrow \{0,1\}^m$. It then produces a vector T with m entries and sets the content of each entry a counter whose initial value is set to 0.
- *Insertion* Given the family of the hash functions $H_1, H_2, \ldots, H_k$, the vector T and an element $e \in S$, if we denote $T[i]$ as the content of the ith entry of T, this algorithm updates the vector T by auto-increasing $T[H_i(e)] := T[H_i(e)] + 1$ for all $i \in \{1, 2, \ldots, k\}$.

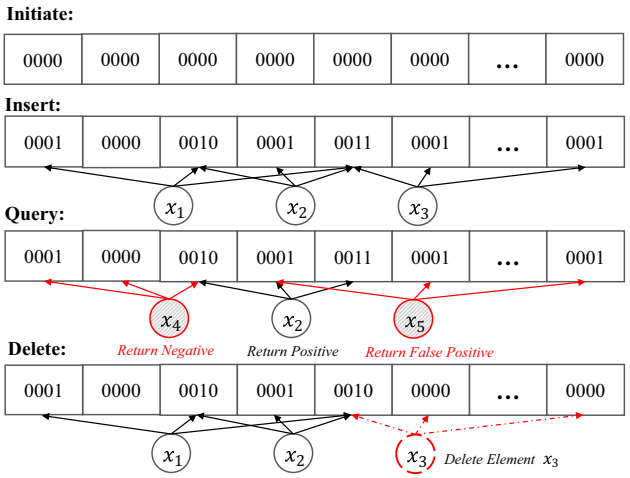

Fig. 2 The working principal of the counting Bloom filter

- *Query* Given the family of the hash functions $H_1, H_2, \ldots, H_k$, the vector T and an element e, the algorithm returns *true* if all $T[H_i(e)] > 0$ for $i \in \{1, 2, \ldots, k\}$; otherwise, it returns *false*.
- *Deletion* Given the family of the hash functions $H_1, H_2, \ldots, H_k$, the vector T and an element $e \in S$, the algorithm delete e from the set S by auto-decreasing $T[H_i(e)] := T[H_i(e)] - 1$ for all $i \in \{1, 2, \ldots, k\}$.

From Fig. 2, we can observe that the Bloom filter may yield false positive. Given the length of the set S, i.e., n, and the number of the entries m, the minimum false positive rate is achieved as $R_f = 2^{-k}$ when $k = \frac{m}{n}\ln 2$ [44].

The Merkle hash tree was first introduced to provide secure authentication using hash functions. As shown in Fig. 3, it's a tree of hashes. The value of leaf nodes is the hash value of the corresponding data node, and every non-leaf node is the hash value of its child nodes' conjunction. For instance, suppose we have a one-way hash function $H(\cdot)$. Then we can compute the value of four leaf nodes by $h_i = H(n_i)$, $(i = 1, 2, 3, 4)$. The value of an internal node has $h_5 = H(H(n_1)||H(n_2))$ and $h_6 = H(H(n_3)||H(n_4))$. Similarly, root node of this tree is computed by $r1 = H(h_5||h_6)$.

3.5 Candidate encoding and result decoding

Candidate encoding A set of candidates, which are denoted by $L_c = \{C_1, C_2, \ldots, C_m\}$, campaign to obtain more votes and compete to win the election. In order to achieving faster tallying process, every candidate's real-world identity is encoded into a digital representation. Inspired by previous work [21,37], we implement the encoding approach as follows.

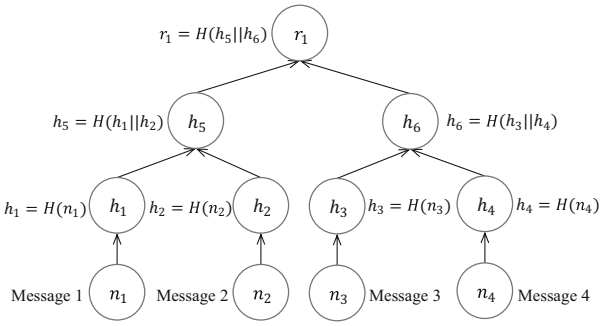

Fig. 3 An example of Merkle hash tree

Assuming the total amount of legitimate voters is $|L_v|$, each candidate cannot obtain more than $|L_v|$ votes. If we denote t_j as the amount of votes that candidate j obtains, for $\forall j \in \{1, 2, 3, \ldots, |L_c|\}$, $t_j < |L_v|$ is always satisfied. Let ℓ be an integer larger than $|L_v|$, possibly where $\ell = |L_v| + 1$, the jth candidate will be encoded as ℓ^{j-1}. For example, the first candidate is encoded as $\ell^0 = 1$, and the last candidate is encoded as $\ell^{|L_c|-1}$. Then voters choice ranges from $\{1, \ell, \ell^2, \ldots, \ell^{|L_c|-1}\}$. As noted in [21], to ensure the correctness of final tally, the parameters are chosen to satisfy $\ell^{|L_c|} < q$, where q is the order of plaintext space of the encryption algorithm.

Result decoding Due to the additive homomorphic property of Paillier encryption scheme, the final tally result R is in form of $R = t_1 + t_2\ell + t_3\ell^2 + \cdots + t_{|L_c|}\ell^{|L_c|-1}$. We define a kind of right shift function $Shift(\cdot)$, where on calling it, the operand will be right shift ℓ position once. This function can be regarded by multiplying the inverse element of ℓ to the operand. For example, $Shift(\ell^n) = \ell^n\ell^{-1} = \ell^{n-1}$. In system implementation, to achieve better performance, the number 2 is always chosen to be the basis of ℓ, where $Shift(\cdot)$ can be directly conducted with right shift instruction on CPU registers.

To decode the tally result, we first apply Extended Euclidean algorithm to compute R_1 and t_1, where $R = R_1\ell + t_1$. Then we put the retrieved R_1 into the defined right shift function to compute $R' = Shift(R_1) = t_2 + t_3\ell^1 + t_4\ell^2 + \cdots + t_{|L_c|}\ell^{|L_c|-2}$. By recursively applying Extended Euclidean algorithm and calling right shift function, we finally get R, R_1, $R_2, \ldots, R_{|L_c|-1}$ and correspondingly $t_1, t_2, t_3, \ldots, t_{|L_c|}$. Respectively, $t_1, t_2, t_3, \ldots, t_{|L_c|}$ are the amount of votes toward each candidate.

4 System model and overview

With the help of diagrams (see Figs. 4, 5), we describe in this section the responsibility of involved entities and provide an overview of Chaintegrity. Beyond that, we define the trust and threat models of Chaintegrity, and explain our design goals.

4.1 Involved entities

As illustrated in Fig. 5, there exists 4 entities in our voting system. Respectively, they are the voters, the election holders, the auditing group and the smart contract.

E1 *Voter* A set of legitimate voters are authorized to vote for their preferred candidates with independent judgments. We assume that all voters hold a long-term public/secret key pair ($\mathsf{pk_V}$, $\mathsf{sk_V}$) that represents their identities.

E2 *Election holders* As the system administrator, election holders are to organize or control the voting process by initializing the system parameters and triggering different phases of an election. To make our system more robust and to defend possible collusion, we assume that n election holders exist in our voting system. These n election holders share a pair of public/secret key pair ("Appendix B.2"). Under this threshold manner, unless at least t election holders are corrupted, malicious adversaries can ruin the voting result.

E3 *Auditing group* An auditing group is an expert panel of auditors who act in the public interest. By checking out all published information, auditing group offers professional opinions on the properness and soundness of elections. To ensure the justness of auditing process, auditors are appointed by different interested parties.

E4 *Smart contract* As the autonomous on-chain code, smart contract will be triggered if the information, which is about to write into a block, includes voter's registration identity (in Preparation and Registration phase), or signatures of encrypted ballot (in Tallying and Opening phase).

A special role named *smart contract administrator* also exists in real-world implementation. With responsibilities to initialize and terminate smart contract, it is always played by an authorized account on blockchain platform (as Ethereum or Hyperledger Fabric). The concrete responsibility of smart contract administrator is slightly different among blockchain platforms. Thus, we will not delve deeper into this role.

As the backbone of our election system, *Blockchain* not only is the runtime host for smart contract, the tamper-resistant bulletin board for verification and auditing, but also provides a low-lever broadcast communication channel [64] for the above involved entities. With the help of blockchain, our election system is more robust and scalable.

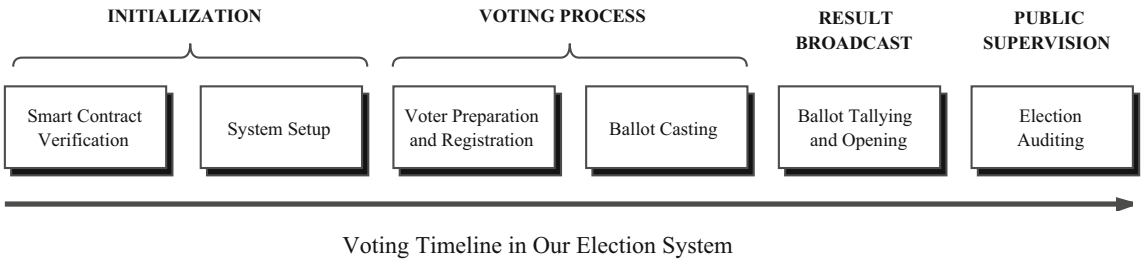

Fig. 4 Six stages exist in our election system

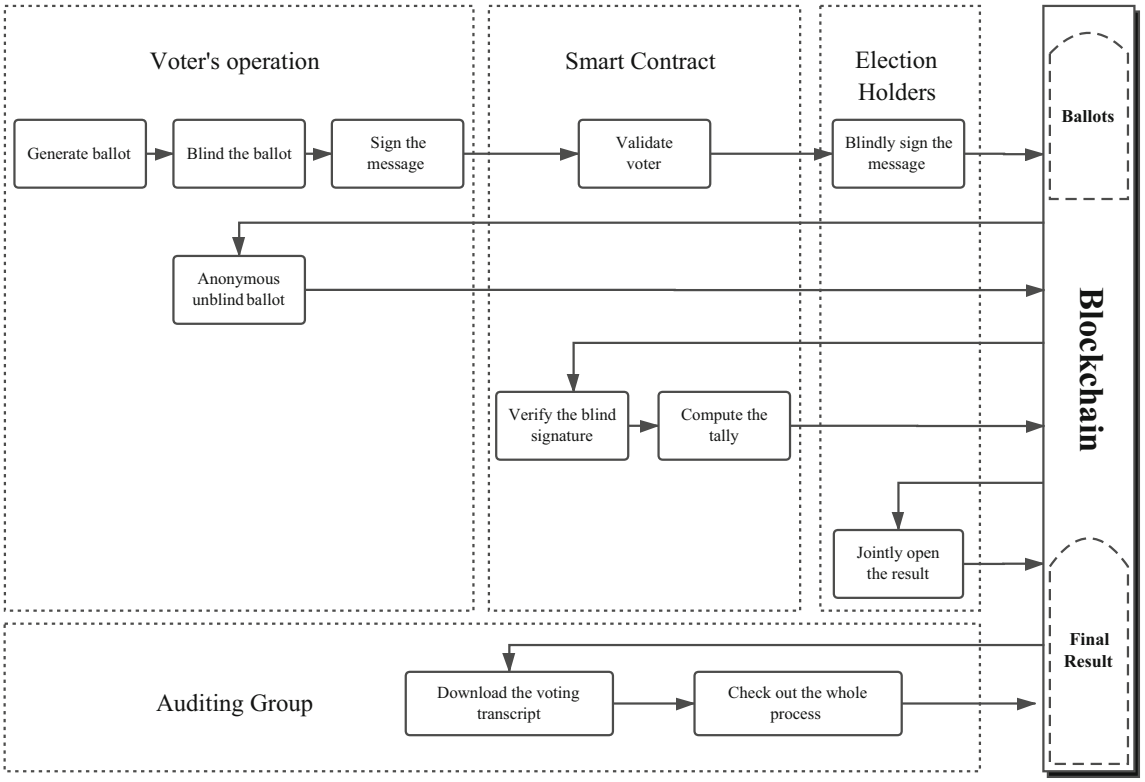

Fig. 5 High-level description: the interactions during the voting process

4.2 System overview

A typical election in our system, as depicted in Fig. 4, includes six distinct stages: Smart Contract Verification, System Setup, Voter Preparation and Registration, Ballot Casting, Ballot Tallying, and Opening and Election Auditing. These six stages are executed chronologically, and triggered autonomously by smart contract. Note that, in a sequence of elections (e.g., multi-level proxy voting), the Smart Contract Verification stage and the System Setup stage are only required to execute once. That is to say, the pair public/secret keys of voters and election holders can be reused in different elections.

The communication flow diagram (see Fig. 5) comprehensively shows the last four stages of our system. To describe what a voter experiences in our election system, we take Alice, a legitimate voter, as an example. Alice only need to interact with the system in stages of Preparation, Registration and Ballot Casting. In Preparation and Registration stage, the voter prepares a ballot and asks for an authorization (registration). And in Ballot Casting stage, voter casts the authorized ballot to the blockchain. After the Ballot Casting phase is over, the system (smart contract) tallies the ballots and publishes the result very efficiently. Since all valid information will be recorded on blockchain immutably, Alice can check the status of her registration request and ballot at any time. Considering those non-technical users (i.e., laymen in digital field), a group of auditors are chosen out in Election Auditing stage to arbitrate the properness of the election.

4.3 Trust and threat models

We assume that most of the validation nodes are *semi-trusted*. That is, the smart contracts will execute honestly, and the blockchain will not equivocate on what is stored; however, the validation nodes may be curious about the voters' privacy. Besides, we make the following assumptions:

- The election holders are all semi-trusted. The election holders cannot be compromised and help adversaries to alter voter choices.[4]
- Our system will not suffer from high coercion risk. Stated differently, we assume that our system is always used in low-coercion elections.
- There exist anonymous communication channels (e.g., the onion router, TOR) so that the voters can be free from IP trace attacks.

As for the threat model, we consider that an computational bounded adversary can launch the following attacks:

- *Replay attack* The adversary may record the previous ballot and fraudulently replay it to the smart contract.
- (*Distributed*) *Denial-of-Service* (*DoS/DDoS*) *Attack* This attack could be either a voter-targeted one or system-targeted one. For the voter-targeted attack, the adversary may block the communication between the target voter and the blockchain; for the other one, the adversary may flood the smart contract with superfluous requests (e.g., registration requests) and disable the normal system functions.
- *Man-in-the-middle attack* [19] The adversary captures and tampers the transmitting message, and then re-transmits it to the smart contract.
- *Sybil attack* [72] The adversary may either attempt to impersonate a legitimate voter to vote or a validation node to attack the blockchain.

4.4 Design goals

To build a system which can be deployed in large-scale elections, our blockchain-enabled voting system should satisfy all the requirements proposed in Sect. 2.1. Apart from this, the system should be secure enough to defend all attacks mentioned above. Also, the system needs to achieve better cost-effectiveness. If we denote n as the total number of the voters, the computational cost of the system should be sub-linear in n, [i.e., $o(n)$] and the communication cost of the system should be linear in n [i.e., $O(n)$].

[4] This assumption is mitigated in Sect. 6.3.

5 Basic e-voting system

In this section, we describe the construction of our blockchain enabled e-voting system and discuss in detail the experience of a voter in a single election contest; extension to multi-contest is trivial. Besides, without loss of generality, we use function notions instead of concrete expression of cryptographic primitives to describe our e-voting system. As for the mathematical constructions of these primitives, interested readers may refer to appendices.

5.1 Prologue of an election

Although this stage seems to be tedious and trivial, a verification of smart contract itself is important and necessary. Considering that the smart contract could be tampered by adversaries and run on malicious validation nodes, a digital fingerprint (a.k.a. file checksums) need to be enclosed with smart contracts and verified by blockchain platform. This verification is carried out in two possible scenarios: at the prologue of an election and when new validation nodes join in.

5.2 System setup

This stage involves the following steps:

Step S1 Election holders jointly generate the global parameter of Paillier cryptosystem $\mathsf{param}_{\mathsf{Paillier}}$ and upload to blockchain. They also jointly generate, and upload the public key $\mathsf{pk}_{\mathsf{Pa_E}}$ of Paillier encryption scheme. The corresponding secret key is split to n parts and each part is held by one election holder ("Appendix B.2").

Step S2 Smart contract administrator runs the setup function of blind signature scheme, and uploads the global parameter $\mathsf{param}_{\mathsf{Blind}}$ to blockchain. From then on, the election is executed autonomously and managed by smart contract.

Step S3 Each election holder generates a pair public/secret key $(\mathsf{sk}_{\mathsf{B}_i}, \mathsf{pk}_{\mathsf{B}_i}) \leftarrow \mathit{BlindGen}(1^k)$, and uploads the public key $\mathsf{pk}_{\mathsf{B}_i}$ to the blockchain.

Step S4 One election holder (chosen by cryptographic sortition) uploads the information of election candidates, which is comprised of their manifesto and encoded identity. Other election holders witness and verify this procedure.

Step S5 Similar as Step S4, a set of encoded identity, with corresponding public key certificate, of legitimate voters is also uploaded to blockchain.

5.3 Voter preparation and registration

A voter, say Alice, first has to use her identity to sign in the voting system. She prepares her ballot and registers in as follows:

Step P1 Alice creates a new account on blockchain. In this phase, she uses this account to interact with smart contract via blockchain.

Step P2 Alice reads the manifestos of each candidate with care, and finds the encoded identity, ℓ^i, of her preferred candidate on blockchain. Then she completes the ballot $x_i = Enc_{\mathsf{pk}_{\mathsf{Pa_E}}}(\ell^i)$.

Step P3 Alice randomly chooses $r_i \in \mathbb{Z}_q$ and blinds the ballot with it $m_i = Blind(x_i, r_i)$.

Step P4 Alice signs the message m_i with her secret key $s_i = Sign_{\mathsf{i}}(m_i)$, and sends $\{m_i, s_i, ID_i\}$ to blockchain. Note that, Alice's signature is generated by normal digital signature scheme.

On receiving Alice's registration request, smart contract operates as follows:

Step R1 Smart contract checks whether or not Alice has applied the registration. This step can be either executed locally or on blockchain. In the local execution manner, smart contract searches for the identity label locally. If Alice is marked as "registered," smart contract rejects Alice's registration; otherwise, it continues the process.[5]

Step R2 Smart contract verifies that whether or not Alice is in the set of legitimate voters. If not, smart contract abolishes Alice's registration; otherwise, it obtains Alice's public key from the public key certificate. Note that, the membership query and the public key query are both executed on blockchain.

Step R3 Smart contract uses Alice's public key to check whether or not s_i is a valid (normal) signature of m_i. If the check fails, smart contract aborts Alice's registration.

Step R4 Smart contract then sends Alice's message m_i to one of the election holders (chosen by cryptographic sortition) via blockchain.

Step R5 The chosen election holder blindly signs on Alice's message m_i as $d_i = BlindSign_{\mathsf{sk}_{\mathsf{B_i}}}(m_i)$, and sends it to smart contract.

Step R6 Smart contract marks Alice as "registered" locally and sends d_i to Alice via blockchain.

[5] Another possible detection on Alice's registration trail is the transaction which smart contract sends back to Alice (in Step R6). This approach is executed on blockchain.

Note that, to confirm that the information is not erasable, Alice needs to wait for 6 block's generation after the block which writes her registration information appears. This process can be conducted by a background program running on Alice's device. As the worst case, she need to witness the generation of 12 blocks (6 for recoding her request and another 6 for recoding smart contract's answer).

5.4 Ballot casting

After the registration of all voters, Alice receives the signature of smart contract and performs the following operations:

Step C1 Alice creates a new account on blockchain. In this phase, she uses this account to interact with smart contract via blockchain.

Step C2 Alice retrieves the desired signature y_i of ballot x_i by unblinding d_i with the previously chosen randomness r_i. She computes $y_i = Unblind(d_i, r_i)$.

Step C3 Alice generates a non-interactive proof of knowledge $\{v_j, e_j, u_j\}_{j \in P} = PoK_{\mathsf{mem}}(x_i, L_c)$ to prove that the plaintext behind ballot x_i is a member of encoded candidates.

Step C4 Alice publishes the ballot/signature pair $\{x_i, y_i\}$ to the blockchain. It will be accepted after the transcript of proof of knowledge $\{v_j, e_j, u_j\}_{j \in P}$ is validated.

5.5 Ballot tallying and opening

For each oncoming block with ballot/signature pair $\{x_i, y_i\}$, the following operations are conducted:

Step T1 Smart contract verifies the ballot/signature pair $\{x_i, y_i\}$ by calling $BlindVerf_{\mathsf{pk}_{\mathsf{B_i}}}(y_i, x_i)$. If the signature is valid, smart contract finds the related block by using block id, and computes the sum of all ballots in this block $x_{B_{id}} = \Sigma_{i=1}^{k}(x_i)$. Otherwise, this ballot/signature pair will be neglected.

After all ballots are collected, smart contract will publish the final encrypted result $x_{all} = \Sigma_{i=1}^{num_{Block}}(x_{B_{id}})$ with a tag "waiting for opening" to the blockchain.

Step O1 Election holders jointly open[6] the result when the *Opening Day* is triggered. They open the tallying result by gathering at least $t + 1$ shared partial

[6] The partial decryption and combination algorithms refer to Appendix B.2. It is also noteworthy that a zero-knowledge proof $(R_1, R_2, e', z) \leftarrow PoK_{cor}(C, C_i)$ generated by each election holder is published to ensure the correctness of the partial decryption.

decryption $C_i = Dec_{\mathsf{sk}_{\mathsf{Pa}_i}}(C)$ and running the combination algorithm $R = Com_{\mathsf{pk}_{\mathsf{Pa}}}(\prod)$, which holds $\prod = \{C_1, C_2, \ldots, C_{t+1}\}$. The result R is published to blockchain.

Step O2 After R is computed, the voting result, $t_1, t_2, t_3, \ldots, t_{|L_c|}$, for each candidate, $\ell^0, \ell^1, \ell^2, \ldots, \ell^{|L_c|-1}$, can be easily obtained, since R is a number to the radix ℓ (see Sect. 3.5 for details).

5.6 Election auditing

This is an optional phase to explicitly show how universal verifiability is achieved. Without this phase, verification and auditing can also be conducted privately and implicitly. Since the communication process is completely recorded and unerasable, everyone can publicly verify the validity of tallying and make sure the integrity of voting process. For an individual voter, say skeptical Bob, he confirms during the election process that, 6 blocks are generated after his messages being writing into a block. Through all his interaction records with smart contract and blockchain, Bob can verify that he casts as intended, the vote is recorded as cast and counted as recorded. Furthermore, an auditing group is chosen to arbitrate controversial elections. By checking the equivalence of the number of ballots and registered voters, and the correctness of the tallying result by decrypting all encrypted ballots, the auditing group can verify and audit the voting process.

6 Extended components

To promote efficiency and robustness, we additionally discuss some components for optimizations and the problems which developers will meet in real-world scenarios. These problems are typical and common in traditional e-voting systems. And we discuss the solutions as system components in the context of blockchain voting.

6.1 Components for fast authentication

Common techniques for anonymous authentication include the ring signature based schemes and the blind signature schemes. In a ring signature-based scheme, the user needs to include a ring of public keys, hide his/her own public key behind it, generate a signature and send to the system. The system only has to verify whether the ring signature is valid. If valid, this implies the public key of the user is among the included ones. Compared with ring signature based schemes, the blind signature-based scheme, as what we described in Sect. 5, requires the system to ensure before issuing the signature that the applicant (i.e., the user) is in the set of the legitimate users.

Unfortunately, as pointed out in [67], searching for a specific transaction and finding out the block with a given transaction are two of the most time-consuming part on the blockchain. Accordingly, the Step R1 and the Step R2 in our voting system will produce large time overhead. If we denote the number of legitimate users as n, the basic authentication algorithm achieves linear time complexity, i.e., $O(n)$. To accelerate the authentication process, an intuitive approach is to load the information of all legitimate voters into the memory. But this will produce large space overhead. Here we propose a hybrid approach to optimize the performance of our system. This approach combines the counting Bloom filter[7] [29] and the Merkle hash tree proof [43] to achieve the $o(logn)$ complexity.

Now we describe the usage of the hybrid data structure. As illustrated in Fig. 6, we build this data structure as follows. We first compute the maximum number $|S|_{\max}$ of the elements that the counting Bloom filter can support with respect to the given false positive rate R_f and the maximum vector length m of the filter. To support as many elements as possible and to remain the minimal false positive rate, we compute $|S|_{\max}$ as follows:

$$|S|_{\max} = -\frac{m}{\ln R_f}(\ln 2)^2. \tag{1}$$

This is derived from the fact that the minimal false positive rate $R_f = 2^{-k}$ is achieved when $k = \frac{m}{n}\ln 2$ [44]. With $|S|_{\max}$, we can compute the Merkle hash tree depth d which is the smallest integer, s.t. $|S|_{\max}2^d \geq n$ where n is the number of supported voters in the system.

Then we can initiate the proposed data structure. We calculate the leaf nodes of the Merkle tree as the hash value of the concatenation of the voter's identity and his/her public key, i.e., $h(ID_i \| \mathsf{pk}_i)$. The value of other tree nodes are calculated as the hash value of the concatenation of its child nodes. As the example in Fig. 6, we calculate all the Merkle trees and we denote by $r_1, r_2, \ldots r_n$ the value of the roots. Then we assign the collection of the value of the roots to the counting Bloom filter $S \leftarrow \{r_1, r_2, \ldots r_n\}$. Then we choose k hash functions and initiate the counting Bloom filter using the method in Sect. 3.4.

Next we describe the registration process. Take Fig. 6 as an example. A voter, e.g., with the identity ID_3 , needs to send the information $\{ID_3, \mathsf{pk}_3, h_4, h_9, m, s\}$ to the system, where s is the signature of m under the corresponding secret key sk_3. Then the system calculates the value of the root node r_1' by using $\{ID_3, \mathsf{pk}_3, h_4, h_9\}$. The system next checks whether or not the calculated r_1' is in the set of root node

[7] In practice, the Cuckoo filters can be regarded as the substitutes of the counting Bloom filters. As for the comparison, one may consult [14,28].

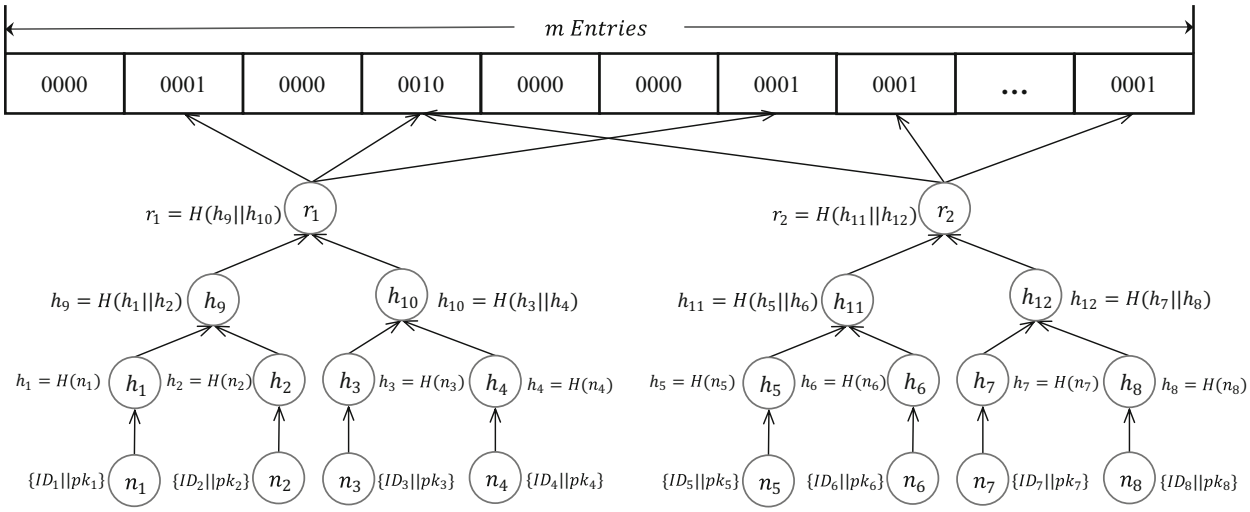

Fig. 6 An example of the hybrid data structure for fast authentication

values S. If the check passes, the system use the pk_3 to verify the authenticity of the signature. If the signature is also valid, the system accepts the registration. Finally, to avoid double registration, the system prunes the Merkle tree and updates the counting Bloom filter.

The proposed mechanism can achieve very fast authentication with a extremely low false positive rate. For example, given the false positive rate $R_f = 2^{-80}$ and the maximum vector length $m = 2\,\text{MB}$, the $|S|_{\max}$ is calculated as $|S|_{\max} = -\frac{m}{\ln R_f}(\ln 2)^2 \approx 43,719$. With the depth $d = 5$ of the Merkle tree, the system can achieve fast authentication among millions of the voters. The time complexity of the proposed mechanism is sub-logarithmic, i.e., $o(\log n)$, since it is the combination of the Bloom filter (constant time algorithm, i.e., $O(1)$) and the Merkle tree [logarithmic time algorithm, i.e., $O(\log n)$].

6.2 Components for cryptographic sortition

We have mentioned the cryptographic sortition in both Step S4 and R4. In our system, sortition helps to decide which election holder is chosen to upload the election information to the blockchain and is responsible for signing a voter's registration. Cryptographic sortition makes our system more fair, as long as the algorithm itself cannot be affected by any entities. However, it is difficult to generate an *agreed random number* in the distributed blockchain setting. One popular way is to use an Oracle [40]. It is a centralized blockchain service provider which accesses outside source and publishes the acquired data on the blockchain to trigger a smart contract. Using an Oracle seems to be efficient and pragmatic, while it breaks the decentralized principal of blockchain and makes our trust model more complicated. Hence, we suggest to utilize block hash, timestamp, or some public information with high entropy as the seed for random number generation [13]. This approach is more robust, and it can be implemented as a smart contract [52] autonomously running on the blockchain.

6.3 Toward end-to-end integrity and robustness

Although Chaintegrity is robust in theory, the real-world deployment may spoil all our efforts. Here is one of the reasons—considering human usability, voters need a mobile application or a computer program to execute operations like ballot casting; however, voters may probably obtain the *wrong* information from their devices [18]. This is mainly caused by two threats: the adversaries may tamper with the client program to display the disguised successful message on voters' devices, and they may use Malware to alter voter choices.

The countermeasures to resist software tampering attack is not complicated. The voting client first needs to be reviewed by the auditing group, and then uploaded to the interplanetary file system (IPFS) [9] which is a peer-to-peer distributed file system as an auxiliary of a blockchain. The integrity of the software is guaranteed by the file fingerprint—a mechanism embedded in IPFS.

To prevent adversaries from altering voter choices, we introduce *code-voting* technique [68] as a component of our system. By adopting this primitive, the voter and the auditing group share secret authentication codes, and the voter needs one more round of interaction with the auditing group in the ballot casting phase. The codes are pre-allocated to the voter by mail service and are secret to election holders or other third parties. In the interaction, the voter exchanges the codes with the auditing group until he/she ensure no adversaries

can falsify his/her preferred candidate choice, and finally he/she sends a lock-in code to fix the record. Code-voting brings three additional benefits to our system. First, it weakens the trust to the election holders and the voting client software. Furthermore, it makes our system robust against voter-targeted DoS attacks. Besides, it provides the authentication codes which can be regarded as the *voter-verified paper audit trail* [42] to achieve end-to-end integrity and to resolve disputes.

7 Security analysis

In this section, we analyze the security of our system by using the aforementioned evaluation criteria. We discuss the typical attacks and the corresponding defence policies. Then we prove the robustness of our system based on the analysis.

7.1 Analysis of evaluation criteria

Here we use the first eight evaluation criteria to assess our proposed system Chaintegrity and leave the analysis of robustness in Sect. 7.3.

Scalability The computational and communication cost of Chaintegrity are respectively sub-linear and linear in the number of voters n, i.e., the $o(n)$ and $O(n)$. This means that our system achieves scalability. The details are discussed in Sect. 8.

Privacy-enhancement The privacy in our system derives from: (a) blind signature makes it impossible to link voter's identity with his/her messages on blockchain; (b) all ballots of the voters are aggregated by the homomorphic property of threshold Paillier encryption; (c) voter uses different accounts in different phases; (d) some additional anonymous communication tools can be utilized to defend IP-trace attacks.

Universal verifiability and end-to-end verifiability The property universal verifiability means any observer can determine whether the election is held faithfully. Chaintegrity abstracts an external role, called the auditing group, to perform the auditing process. As described in Sect. 5.6, the auditing group can verify the completeness of the election executed via Chaintegrity.

Rather, the end-to-end verifiability requires the system to allow the voters to verify that he/she casts as intended, the vote is recorded as cast and counted as recorded. The inherent feature of blockchain makes sure that the vote is recorded as cast and counted as recorded. The voters can use the block index and the transaction index to verify this. The *cast-as-intended* sub-property is guaranteed by integrating code-voting technique in our system. In Sect. 6.3, we have discussed this procedure.

Vote-and-go In Chaintegrity, the voters have no further operations after they ensure that their ballots are recorded on blockchain. This feature matches the requirement of vote-and-go.

Unreusability Repeated ballot/signature pair can be intuitively detected on blockchain and will only be counted once. To cast more than one accepted ballot, voter has to send two identical ballot/signature pair. In registration process, he/she can only get one official signature with his/her identity. Accordingly, he/she need to forge another valid ballot/signature pair, which contradicts to the unforgeability property of the blind signature scheme.

Affordability Chaintegrity does not utilize any cryptocurrency incentives. Election holders need not to pay for the rewards. By applying consortium blockchain, the transaction fees are as well largely reduced or eliminated.

Fairness Ballots are tallied and opened after the termination of Ballot Casting phase, and they are encrypted under the public key of threshold Paillier encryption system. To decrypt the partial result in advance, there must exists at least $t + 1$ corrupted election holders. However, this violates our assumptions.

7.2 Typical attacks and defence

We list four typical attacks that are common in the e-voting systems and the distributed setting. At the same time, we discuss our defence strategies. This is not a exhaustive list, but reflects our design rationale.

Replay attack As a form of attack over network communication channel, replay attack is conducted by either the originator or adversaries to abuse system source [20]. Replay attack can be regarded as a special Man-in-the-Middle attack. In the context of election system, malicious voter and adversaries attempt to vote more than once by replaying the valid ballot/signature pair. However, by adopting double voting detection strategy (a.k.a. unreusability), our system can always defend this type of attack.

(Distributed) Denial-of-Service (DoS/DDoS) attack DoS/DDoS attack is commonly conducted in centralized election systems. By disrupting centralized voting service or tallying process, the election result may be inaccurate to reflect popular preference. Even worse, in some decentralized elections as [35,41], DoS/DDoS attacks, whose target is (even

one) legitimate voter, can cause fatal errors and processes-crashing in final tallying. Our system will not suffer from outage, unless all blockchain validation nodes are corrupted. As for the voter-targeted DoS attack, we discuss the defensive policy as an extension in Sect. 6.3.

Man-in-the-middle attack In public communication environment, man-in-the-middle attack describes the process that attacker relays and alters the communication between two parities (voter and system, in the context of e-voting). In the model of blockchain voting, every voting message is checked or signed before written into a block. If attackers forge on signatures or flip the transmitted message, it will not pass the validation of smart contract. On the perspective of voter, he/she will can never know the reason that his/her message is not written into blockchain is caused by network failure or interpolation. But he/she can simply resend his/her message after timeout, unless it appears on blockchain.

Sybil attack Sybil attack [25] is a well-known attack in peer-to-peer network or decentralized architecture, where attackers disrupt the network by creating a large number of pseudonymous (accounts) to gain a overwhelm advantage over normal users. Sybil attack could influence blockchain network itself and distributed applications. Permissioned blockchain copes this problem with an authentication mechanism when new validation node joining in, while permissionless blockchain suggests different Sybil-resistant approaches such as mining and cryptographic sortition. Above the blockchain platform, our distributed election system also maintains strong resistance to Sybil attack. As guaranteed by voter authentication, only legitimate voters can get the right to cast their vote. As protected by unforgeability of blind signature, only authenticated ballots will be tallied. Hence Sybil attackers cannot use pseudonymous to ruin our election system.

7.3 Robustness analysis

To prove the robustness of the system, we first should prove the resilience of Chaintegrity to defend the typical attacks, which is presented in Sect. 7.2. Then we need to prove the partial failure tolerance and adversarial input resistance of Chaintegrity. Chaintegrity is built upon blockchain which eliminates the centralized sever. As long as most validation nodes execute truthfully, Chaintegrity will remain normality. With regards the resistance to the adversarial input, the requests which are not matched with the preset rules in the smart contract will simply be discarded. Conclusively, we have to prove the ability of the system to run the election when confronting a minority of dishonest election authorities. Dishonesty of the election holders may lead to two circumstances—the rejection to decrypt the tallying result and the deception of providing wrong information to the voters. The first circumstance will not effect Chaintegrity, due to the threshold property and our assumptions. As for the second one, the code-voting component in Chaintegrity can provide an effective protection. Hence we proved the robustness of the system.

Table 2 Time consumption of cryptosystems

Cryptosystems	Algorithms	Notation	Time (ms)
Paillier	Encryption	T_{Pa_Enc}	0.001299
	Decryption	T_{Pa_Dec}	0.000384
	Addition	T_{Pa_Add}	1.49557e−05
Schnorr	Signature	T_{Sch_Sig}	0.054979
	Verification	T_{Sch_Vfy}	0.109590
Okamoto–Schnorr	Blind	T_{Blind}	1.23500e−06
	Signature	T_{Blind_Sign}	1.23500e−06
	Unblind	$T_{Unblind}$	0.017206
	Verification	T_{Blind_Verf}	0.107307

8 Performance evaluation

In this section, we discuss the efficiency of the proposed voting scheme. The performance analysis is based on the computational cost of three main processing steps, which are Voter Preparation and Registration, Ballot Casting and Ballot Tallying and Opening.

To clearly describe the computational efficiency, we implemented the experiment using the gmpy2 python module and deployed our system on a computer equipped with an Intel Core i7-7700 processor with a speed of 3.6 GHz and a memory of 8 GB. In our scheme, we used the 1024-bit keys for the Paillier cryptosystem and the blind signature. All the computational time of our employed cryptographic processes was enumerated in Table 2. Then we analyze the efficiency performance in each phase from the perspective of voter experience and system concurrency.

8.1 Voter preparation and registration performance

From the perspective of voter, in one round of preparation and registration, voter needs to generate a ciphertext of his/her preferred candidate, blind the ballot and make a signature to prove his/her identity. Thus, we use T_{voter} to denote the time spent of a voter during this step. This variable is presented as the equation bellow. We substitute the concrete time benchmark from Table 2 into the equation, and get a proximate platform-dependent time spent.

$$T_{prep} = T_{Pa\_Enc} + T_{Blind} + T_{Sch\_Sig} = 0.056279235 \text{ (ms)}. \quad (2)$$

In this phase, we assumed that there are n votes. T_{reg} denotes the time consumption of smart contract and election holder, while the time consumption of the search algorithm is not included. Then T_{reg} can be presented as the following equation.

$$T_{reg} = (T_{Sch\_Vfy} + T_{Blind\_Sign}) \times n = 0.109591235 \times n \text{ (ms)}. \quad (3)$$

8.2 Ballot casting performance

In this phase, only voters are involved. T_{PoK} denotes the time spent on generation of the membership Proof of Knowledge. Then we use the T_{ballot} to denote the ballot time that a voter spent, which can be presented as:

$$T_{ballot} = T_{Unblind} + T_{PoK} = 0.017206 + T_{PoK} \text{ (ms)}. \quad (4)$$

8.3 Ballot tallying and opening performance

System efficiency can be measured in two dimensions: the voter experience and the concurrency of smart contract. In the blockchain-based systems, voter needs to wait the generation of six blocks to confirm that his/her ballot is not be tampered. Then, the T_{tally} is used to denote the total time spent tallying and opening the result, which is presented as:

$$T_{tally} = T_{Blind\_Verf} \times n + T_{Pa\_Add} \times n + T_{Pa\_Dec} = 0.1073219557 \times n + 0.000384 \text{ (ms)}. \quad (5)$$

8.4 Overall performance

Suppose there are n voters involved, and s denotes the length of one voter's communication overhead in our system. Then we can compute the communication cost of one ballot as $O(s \times n)$. The number of voters is much larger than the number of one voter's communication cost, which is $n \gg s$. Therefore the communication cost of our system can be regarded as linear with the number of voters. The upper bound of the proposed voting system's communication complexity can be expressed as $O(n)$, which leads to a cost-effective system. The efficiency analysis shows that the proposed system is suitable for the large-scale voting situations.

9 Conclusion and future work

In this paper, we have proposed a new blockchain enabled voting system, Chaintegrity, for large-scale elections that fulfills cost-effectiveness, verifiability, robustness and other properties. The system is deployed as smart contracts executing on the blockchain. Some cryptographic primitives, such as blind signature and threshold Paillier encryption, are tailored and adapted in Chaintegrity to backup such properties. Further, we introduced some extended components to make Chaintegrity more efficient and robust. A hybrid component that combines the counting Bloom filter and the Merkle hash tree is also proposed to achieve fast authentication.

Despite this advance, further investigation into the other electoral systems via blockchain is a direction for future work. Chaintegrity, with other existing blockchain enabled voting systems, focuses on plurality electoral systems. Other types of electoral systems, like quadratic voting [49] and statement voting [70], are becoming hot topics in this field. Hence it could be an intriguing open problem to discuss the implementation of these systems via blockchain.

Acknowledgements We thank the anonymous reviewers for their invaluable comments and suggestions. This work was supported in part by the 13th Five-Year Plan of National Cryptography Development Fund for Cryptographic Theory of China under Grant MMJJ20170204, in part by the Fundamental Research Funds for the Central Universities under Grant ZYGX2016J091, the Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, and in part by the Natural Science Foundation of China under Grants U1401257, 61472064, and 61602096, Sichuan Science and Technology Project under Grant 2018KZ007.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix A: Digital signature

Appendix A.1: Schnorr signature

In this signature scheme [54], p and q are primes, which $q|p-1$, $q \geqslant 2^{140}$, $p \geqslant 2^{512}$. An integer $t = O(|p|)$ (e.g., $t \geq 20$), and let g be a generator of a multiplicative subgroup of Z_p with order q. m is the message to be signed. Let s denotes the Signer's private key, which is a random number chosen by the Signer in $\{1, 2, \ldots, q\}$. And v denotes the Signer's corresponding public key, which is the number $v = g^{-s} \bmod p$.

Common input: G, Z_q, public key (p, q, g, t, v), $\mathscr{H} : G \times Z_q \to Z_{2^t}$.

Signer's private input: $s \in Z_q$.
Sign Phase. The Signer:

- Chooses a random number $k \in_R Z_q^*$.
- Computes $r = g^k$, $e = \mathscr{H}(r, m)$ and $y = se + k \bmod q$.

Then the (e, y) is the Signer's signature of message m.
Verification Phase. The Verifier:

- Computes $x = g^y v^e \bmod q$
- Checks if $e \stackrel{?}{=} \mathscr{H}(x, m)$.

Appendix A.2: Okamoto–Schnorr blind signature

In practice, we use the Okamoto–Schnorr blind signature [46]. In this blind signature scheme, all the parameters are the same as the Schnorr Signature above. And the details of this blind signature scheme are as follow:

Common input: G, Z_q, public key (p, q, g, t, v), $\mathscr{H} : G \times Z_q \to Z_{2^t}$.
Signer's private input: $s \in Z_q$.
User's private input: m.
Commit Phase. The Signer:

- Picks a random number $r \in Z_q$.
- Computes $x = g^r \bmod p$.
- Sends x to the User.

Blind Phase. The User:

- Picks two random numbers $d, u \in Z_q$.
- Computes $x^* = g^u v^{-d} x \bmod p$, $e^* = \mathscr{H}(x^*, m)$, $e = e^* + d \bmod q$.
- Sends e to the Signer.

Sign Phase. The Signer:

- Computes $y = r + es \bmod q$.
- Sends y to the User.

Unblind Phase. The User:

- Computes $y^* = y + u \bmod q$.

Then the (x^*, e^*, y^*) is the Signer's signature of message m.

Verification Phase. The Verifier:

- Computes $e^{*'} = \mathscr{H}(x^*, m)$.
- Checks if $x^* = g^{y^*} v^{e^{*'}}$ and $e^{*'} = e^*$.

Remark The Okamoto–Schnorr blind signature requires one more round, i.e., the commit phase, than the original blind signature. In the adaptation to our scheme, all election holders generate such a commitment to the legitimate voters and allocate to them in advance. In the authentication process, the voter triggers the sortition smart contract to choose one specific election holder. Then the voter selects the corresponding commitment to blind the ballot and then sends the blinded message to the smart contract. Other steps will follow the protocol described in the main part of our paper.

Appendix B: Homomorphic encryption

Appendix B.1: Paillier encryption

In our system, we use the Paillier encryption scheme [48] to achieve homomorphic encryption, and the details of this scheme are as follow.

In this encryption scheme, choose two prime numbers p and q, which $gcd(p, q-1) = gcd(p-1, q) = 1$. Then $\lambda = \text{lcm}(p-1, q-1)$ and $N = p \cdot q$. Define $L(b) = \frac{b-1}{N}$, where $b \in Z_{N^2}^*$. Choose a random element g, where $g \in Z_{N^2}^*$. Compute $\mu = (L(g^\lambda \bmod N^2))^{-1} \bmod N$. The public key is (N, g) and the secret key is (λ, μ, p, q). Then let $\ell \in Z_N$ be the plaintext. To encrypt the plaintext, select a random number $r \in Z_N^*$ and compute the ciphertext $C = g^\ell r^N \bmod N^2$. To decrypt the ciphertext, compute $\ell = (L(C^\lambda \bmod N^2) \cdot \mu) \bmod N$.

Additive Homomorphic Property

For anyone who has the public key and the different ciphertexts $c_1 = g^{\ell_1} r_1^N \bmod N^2$ and $c_2 = g^{\ell_2} r_2^N \bmod N^2$ of plaintexts ℓ_1 and ℓ_2 from different users, the encryption of $\ell_1 + \ell_2$ is easy to generate by $c_1 \cdot c_2 = g^{\ell_1 + \ell_2} r_1^N r_2^N \bmod N^2$.

For ρ users, the encryption of $\Sigma_{i=1}^{\rho} \ell_i$ can be generated by $\Pi_{i=1}^{\rho} c_i = \Pi_{i=1}^{\rho} g^{\ell_i} r_i^N \bmod N^2$. To decrypt the ciphertext, compute

$$\Sigma_{i=1}^{\rho} \ell_i = (L((\Pi_{i=1}^{\rho} c_i)^\lambda \bmod N^2) \cdot \mu) \bmod N.$$

Appendix B.2: Threshold version of Paillier encryption

Suppose there are n parties sharing the secret together. And if there are fewer than $t+1$ valid partial decryption shares of

the parties, the ciphertext cannot be decrypted. The parties execute the distributed RSA modulus generation protocol and the key generation algorithm in [12,45].

After the algorithm is successfully executed, the public key (N, g) is published with an agreed global parameter θ which is used to combine partial ciphertexts. Each party P_i gets a share of secret key which is the polynomial $f(i)$. Also, P_i generates and distributes a verification key $VK_i = v^{\Delta f(i)} \bmod N^2$ where $v \in_R Q_{N^2}$. The verification key is used to proof the correctness of partial decryption (see "Appendix B.3").

Let ℓ be the plaintext, and the ciphertext is generated by $C = g^{\ell} r^N \bmod N^2$. Then, the following steps can be performed by any $t+1$ parties to decrypt the message:

1. *Decryption* Each party P_i generates and shares the partial decryption $C_i = C^{2\Delta f(i)} \bmod N^2$ where $\Delta = n!$.
2. *Combination* Define $L(u) = \frac{u-1}{N}$, $\lambda^S_{x,i} = \Pi_{i' \in S \setminus \{i\}} \frac{x-i'}{i-i'}$ and $\mu_i = \Delta \times \lambda^S_{0,i} \in \mathbb{Z}$. And the message can be recovered through

$$\ell = L(\Pi_{i \in S} C_i^{2\mu_i} \bmod N^2) \times \frac{1}{-4\Delta^2\theta} \bmod N.$$

Appendix B.3: Zero-knowledge proofs

Non-interactive zero-knowledge proof of membership [8]

In this section, an efficient non-interactive proof of knowledge scheme is described as follows. If Alice has a ciphertext c of the message m which is in a set of n plaintext. She can use this scheme to prove that the ciphertext c is from one of n plaintext in a set.

Let N be the RSA modulus of Paillier encryption system. Define $\gamma = \{\ell_1, \ell_2, \ldots, \ell_\rho\}$ as the set of ρ encoded candidates. Let P denote the set of n messages and C denote the ciphertext. And g is the public key in the Paillier encryption scheme. Furthermore, we define that $a \div b$ equals the quotient in the division of a by b. In this proof, the Prover and the Verifier are involved.

Common input: N, $\gamma = \{\ell_1, \ell_2, \ldots, \ell_\rho\}$, P, C, g, $\mathscr{H}' : Z^*_{N^2} \to \{0,1\}^k (k \geq 80)$.
Prover's private input: $r \in Z^*_N$.
Proof Phase. The Prover:

- Choose a random number $\kappa \in Z^*_N$.
- Choose $\rho - 1$ random numbers $\{e_j\}_{j \neq i} \in Z_N$, and $\rho - 1$ random numbers $\{v_j\}_{j \neq i} \in Z^*_N$.
- Compute $u_i = \kappa^N \bmod N^2$.
- Compute $\{u_j = v_j^N (g^{\ell_j}/C)^{e_j} \bmod N^2\}_{j \neq i}$.
- Calculate the hash value $e = \mathscr{H}'(\Sigma_j u_j)$.
- Let $e_i = e - \sum_{j \neq i} e_j \bmod N$
- Calculate $v_i = \kappa \cdot r^{e_i} \cdot g^{(e - \Sigma_{j \neq i} e_j) \div N} \bmod N$
- Send $\{v_j, e_j, u_j\}_{j \in P}$ to the Verifier.

Verification Phase. The Verifier:

- Calculate the hash value $e = \mathscr{H}'(\Sigma_j u_j)$.
- Check if $e = \mathscr{H}'(\Sigma_j u_j)$ and $v_j^N = u_j (C/g^{\ell_j})^{e_j} \bmod N^2$ for each $j \in P$.

Non-interactive zero-knowledge proof of correctness of partial decryption [23]

In this section, a non-interactive zero-knowledge proof of correctness of partial decryption scheme is described as follows. If Alice decrypts the ciphertext c to get the partial decryption message c_i. She can use this scheme to prove that the partial decryption message c_i is decrypted correctly with her partial private key.

The party P_i takes $f(i), v, VK_i = v^{\Delta f(i)}, C \in Z_{N^2}$ as input, and the partial decryption $C_i = C^{2\Delta f(i)} \bmod N^2$ is generated. Then the zero-knowledge proof protocol is executed to prove the equality that $f(i) = \log_{C^{4\Delta}}(C_i)^2 = \log_{v^{\Delta}} VK_i$. The steps of this non-interactive proof are as follows:

Common input: v, VK_i, C, C_i, $\mathscr{H}'' : \{0,1\}^* \to \{0,1\}^k (k \geq 80)$.
Party's private input: $f(i)$.
The Parties:

- Choose a random number $r \in_R [0, 2^{2k}T]$, where T is the upper bound of $f(i)$ and k is the security parameter which holds $k \geq 80$.
- Generate $R_1 = v^{\Delta r} \bmod N^2$ and $R_2 = C^{4\Delta r} \bmod N^2$.
- Send R_1 and R_2 to the verifier.
- Choose $e' = \mathscr{H}''(C, C_i, v, VK_i, R_1, R_2)$ and send e' to P_i.
- Generate $z = r + e' f(i)$ in Z and send z to the verifier.

The Verifier:

- Checks if $v^{\Delta z} \equiv R_1 (VK_i)^{e'} \bmod N^2$ and $C^{4\Delta z} \equiv R_2 C_i^{2e'} \bmod N^2$.

References

1. Adida, B.: Helios: web-based open-audit voting. In: USENIX Security Symposium, vol. 17, pp. 335–348 (2008)
2. Agora: Bringing voting systems into the digital age. https://www.agora.vote/. Accessed 30 March 2019
3. Alvarez, R.M., Levin, I., Li, Y.: Fraud, convenience, and e-voting: how voting experience shapes opinions about voting technology. J. Inf. Technol. Polit. **15**(2), 94–105 (2018)
4. Alves, J., Pinto, A.: On the use of the blockchain technology in electronic voting systems. In: International Symposium on Ambient Intelligence, pp. 323–330. Springer, Berlin (2018)
5. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, p. 30. ACM (2018)
6. Bajak, F.: Apnewsbreak: Georgia election server wiped after suit filed. https://apnews.com/877ee1015f1c43f1965f63538b035d3f. Accessed 30 March 2019
7. Bartolucci, S., Bernat, P., Joseph, D.: Sharvot: secret share-based voting on the blockchain. In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, pp. 30–34. ACM (2018)
8. Baudron, O., Fouque, P.A., Pointcheval, D., Stern, J., Poupard, G.: Practical multi-candidate election system. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, pp. 274–283. ACM (2001)
9. Benet, J.: IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561 (2014)
10. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Annual Cryptology Conference, pp. 421–439. Springer, Berlin (2014)
11. Bistarelli, S., Mantilacci, M., Santancini, P., Santini, F.: An end-to-end voting-system based on Bitcoin. In: Proceedings of the Symposium on Applied Computing, pp. 1836–1841. ACM, New York (2017)
12. Boneh, D., Franklin, M.: Efficient generation of shared RSA keys. In: Annual International Cryptology Conference, pp. 425–439. Springer, Berlin (1997)
13. Bonneau, J., Clark, J., Goldfeder, S.: On Bitcoin as a public randomness source. IACR Cryptology ePrint Archive 2015, p. 1015 (2015)
14. Breslow, A.D., Jayasena, N.S.: Morton filters: faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. Proc. VLDB Endow. **11**(9), 1041–1055 (2018)
15. Cachin, C.: Architecture of the hyperledger blockchain fabric. In: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, vol. 310 (2016)
16. Chaieb, M., Yousfi, S., Lafourcade, P., Robbana, R.: Verify-your-vote: a verifiable blockchain-based online voting protocol. In: European, Mediterranean, and Middle Eastern Conference on Information Systems, pp. 16–30. Springer, Berlin (2018)
17. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology, pp. 199–203. Springer, Berlin (1983)
18. Chaum, D., Essex, A., Carback, R., Clark, J., Popoveniuc, S., Sherman, A., Vora, P.: Scantegrity: end-to-end voter-verifiable optical-scan voting. IEEE Secur. Priv. **6**(3), 40–46 (2008)
19. Chen, C.M., Wang, K.H., Yeh, K.H., Xiang, B., Wu, T.Y.: Attacks and solutions on a three-party password-based authenticated key exchange protocol for wireless communications. J. Ambient Intell. Human. Comput. **10**(8), 3133–3142 (2018)
20. Chen, C.M., Xiang, B., Liu, Y., Wang, K.H.: A secure authentication protocol for internet of vehicles. IEEE Access **7**, 12047–12057 (2019)
21. Chow, S.S., Liu, J.K., Wong, D.S.: Robust receipt-free election system with ballot secrecy and verifiability. In: NDSS, vol. 8, pp. 81–94 (2008)
22. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security, pp. 106–125. Springer, Berlin (2016)
23. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 152–165. Springer, Berlin (2001)
24. DeMuro, J.: Here are the 10 sectors that blockchain will disrupt forever. https://www.techradar.com/news/here-are-the-10-sectors-that-blockchain-will-disrupt-forever. Accessed 30 March 2019
25. Douceur, J.R.: The Sybil attack. In: International Workshop on Peer-to-Peer Systems, pp. 251–260. Springer, Berlin (2002)
26. EOSIO: EOS.IO technical white paper v2. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. Accessed 30 March 2019
27. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper. Accessed 30 March 2019
28. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, pp. 75–88. ACM, New York (2014)
29. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000)
30. FollowMyVote: The online voting platform of the future. https://followmyvote.com/. Accessed 30 March 2019
31. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: International Workshop on the Theory and Application of Cryptographic Techniques, pp. 244–251. Springer, Berlin (1992)
32. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 281–310. Springer, Berlin (2015)
33. Gibson, J.P., Krimmer, R., Teague, V., Pomares, J.: A review of e-voting: the past, present and future. Ann. Telecommun. **71**(7–8), 279–286 (2016)
34. Gramoli, V.: From blockchain consensus back to byzantine consensus. In: Future Generation Computer Systems (2017)
35. Hao, F., Ryan, P.Y., Zieliński, P.: Anonymous voting by two-round public discussion. IET Inf. Secur. **4**(2), 62–67 (2010)
36. Heiberg, S., Kubjas, I., Siim, J., Willemson, J.: On trade-offs of applying block chains for electronic voting bulletin boards. In: E-Vote-ID 2018, p. 259 (2018)
37. Jiang, Q., Huang, X., Zhang, N., Zhang, K., Ma, X., Ma, J.: Shake to communicate: secure handshake acceleration-based pairing mechanism for wrist worn devices. IEEE Internet Things J. **6**(3), 5618–5630 (2019)
38. Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., Ford, B.: Omniledger: a secure, scale-out, decentralized ledger via sharding. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 583–598. IEEE (2018)
39. Kshetri, N., Voas, J.: Blockchain-enabled e-voting. IEEE Softw. **35**(4), 95–99 (2018)
40. Ltd., O.: Oraclize documentation. https://docs.oraclize.it/. Accessed 30 March 2019
41. McCorry, P., Shahandashti, S.F., Hao, F.: A smart contract for boardroom voting with maximum voter privacy. In: International

Conference on Financial Cryptography and Data Security, pp. 357–375. Springer, Berlin (2017)
42. Mercuri, R.T.: On auditing audit trails. Commun. ACM **46**(1), 17–20 (2003)
43. Merkle, R.C.: Protocols for public key cryptosystems. In: 1980 IEEE Symposium on Security and Privacy, pp. 122–122. IEEE (1980)
44. Mitzenmacher, M.: Compressed bloom filters. IEEE/ACM Trans. Netw. **10**(5), 604–612 (2002)
45. Nishide, T., Sakurai, K.: Distributed Paillier cryptosystem without trusted dealer. In: International Workshop on Information Security Applications. pp. 44–60. Springer, Berlin (2010)
46. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Annual International Cryptology Conference, pp. 31–53. Springer, Berlin (1992)
47. Okamoto, T.: Receipt-free electronic voting schemes for large scale elections. In: International Workshop on Security Protocols, pp. 25–35. Springer, Berlin (1997)
48. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques, pp. 223–238. Springer, Berlin (1999)
49. Park, S., Rivest, R.L.: Towards secure quadratic voting. Public Choice **172**(1–2), 151–175 (2017)
50. Pawlak, M., Guziur, J., Poniszewska-Marańda, A.: Voting process with blockchain technology: auditable blockchain voting system. In: International Conference on Intelligent Networking and Collaborative Systems, pp. 233–244. Springer, Berlin (2018)
51. Qin, Z., Sun, J., Wahaballa, A., Zheng, W., Xiong, H., Qin, Z.: A secure and privacy-preserving mobile wallet with outsourced verification in cloud computing. Comput. Stand. Interfaces **54**, 55–60 (2017)
52. RANDAO: RANDAO: a DAO working as RNG of Ethereum. https://github.com/randao/randao/blob/master/README.md. Accessed 30 March 2019
53. Ryan, P.Y., Bismark, D., Heather, J., Schneider, S., Xia, Z.: Prêt à voter: a voter-verifiable voting system. IEEE Trans. Inf. Forensics Secur. **4**(4), 662–673 (2009)
54. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
55. Scott, D.: North Carolina elections board orders new house election after ballot tampering scandal. https://www.vox.com/policy-and-politics/2019/2/21/18231981/north-carolina-election-fraud-new-nc-9-election. Accessed 30 March 2019
56. Takabatake, Y., Kotani, D., Okabe, Y.: An anonymous distributed electronic voting system using Zerocoin (2016)
57. Tian, H., Fu, L., He, J.: A simpler Bitcoin voting protocol. In: International Conference on Information Security and Cryptology, pp. 81–98. Springer, Berlin (2017)
58. TIVI: TIVI powered by smartmatic and cybernetica—tivi.io. https://tivi.io/. Accessed 30 March 2019
59. Wang, K.H., Mondal, S.K., Chan, K., Xie, X.: A review of contemporary e-voting: requirements, technology, systems and usability. Data Sci. Pattern Recogn. **1**(1), 31–47 (2017)
60. Xiong, H.: Cost-effective scalable and anonymous certificateless remote authentication protocol. IEEE Trans. Inf. Forensics Secur. **9**(12), 2327–2339 (2014)
61. Xiong, H., Qin, Z.: Revocable and scalable certificateless remote authentication protocol with anonymity for wireless body area networks. IEEE Trans. Inf. Forensics Secur. **10**(7), 1442–1455 (2015)
62. Xiong, H., Mei, Q., Zhao, Y.: Efficient and provably secure certificateless parallel key-insulated signature without pairing for IIoT environments. IEEE Syst. J. (2018). https://doi.org/10.1109/JSYST.2018.2890126
63. Xiong, H., Zhang, H., Sun, J.: Attribute-based privacy-preserving data sharing for dynamic groups in cloud computing. IEEE Syst. J. (2018). https://doi.org/10.1109/JSYST.2018.2865221
64. Xiong, H., Zhao, Y., Peng, L., Zhang, H., Yeh, K.H.: Partially policy-hidden attribute-based broadcast encryption with secure delegation in edge computing. Future Gener. Comput. Syst. **97**, 453–461 (2019)
65. Yaga, D., Mell, P., Roby, N., Scarfone, K.: Blockchain Technology Overview. Technical report, National Institute of Standards and Technology (2018)
66. Yang, X., Yi, X., Nepal, S., Han, F.: Decentralized voting: a self-tallying voting system using a smart contract on the Ethereum blockchain. In: International Conference on Web Information Systems Engineering, pp. 18–35. Springer, Berlin (2018)
67. Yu, B., Liu, J.K., Sakzad, A., Nepal, S., Steinfeld, R., Rimba, P., Au, M.H.: Platform-independent secure blockchain-based voting system. In: International Conference on Information Security, pp. 369–386. Springer, Berlin (2018)
68. Zagórski, F., Carback, R.T., Chaum, D., Clark, J., Essex, A., Vora, P.L.: Remotegrity: design and use of an end-to-end verifiable remote voting system. In: International Conference on Applied Cryptography and Network Security, pp. 441–457. Springer, Berlin (2013)
69. Zhang, H., Deng, E., Zhu, H., Cao, Z.: Smart contract for secure billing in ride-hailing service via blockchain. Peer-to-Peer Netw. Appl. **12**(5), 1346–1357 (2019)
70. Zhang, B., Zhou, H.S.: Statement voting. In: Financial Cryptography and Data Security 2019 (2018)
71. Zhao, Z., Chan, T.H.H.: How to vote privately using Bitcoin. In: International Conference on Information and Communications Security, pp. 82–96. Springer, Berlin (2015)
72. Zheng, H., Xue, M., Lu, H., Hao, S., Zhu, H., Liang, X., Ross, K.W.: Smoke Screener or Straight Shooter: Detecting Elite Sybil Attacks in User-Review Social Networks, NDSS (2018)